



Programming Support Library (PSL)
Technical Report

AD A107250

A107249

LEVEL

7/1

DTIC FILE COPY

DTIC
ELEC
NOV 17 1981
S
A

IBM

81 11 10 004

REPORT DOCUMENTATION PAGE		1. REPORT NO. 18 DOD/DF 81/016b		2. Recipient's Accession No. AD-A107 250	
4. Title and Subtitle 6 PROGRAMMING SUPPORT LIBRARY (PSL), Technical Report				11. Date May 1978	
7. Author(s)				8. Performing Organization Rept. No. 1251	
9. Performing Organization Name and Address Federal Systems Division International Business Machines Corporation Gaithersburg, Maryland				10. Project/Task/Work Unit No.	
12. Sponsoring Organization Name and Address Rome Air Development Center RADC/COEE Griffiss Air Force Base, NY 13441				11. Contract(G) or Grant(G) No. 15 F38602-77-C-0249	
13. Supplementary Notes 2 For magnetic tape, see AD-A107/248 See also AD-A107 251.				13. Type of Report & Period Covered	
15. Abstract (Limit: 200 words) The Programming Support Library (PSL) is a software system which provides the tools to organize, implement, and control computer program development. This involves the support of the actual programming process and also the support of the management process. The PSL is designed to support Top Down Design and Structured Programming (TDDSP).				14.	
<p style="text-align: right;">DTIC SELECTED NOV 17 1981</p>					
27. Document Analysis a. Descriptors Software Configuration Control Structured Programming Support Tool Software Development Support					
b. Identifiers/Open-Ended Terms					
c. COSATI Field/Group 174950					
18. Availability Statement:				19. Security Class (This Report) UNCLASSIFIED	
				20. Security Class (This Page) UNCLASSIFIED	
				21. No. of Pages	
				22. Price	

DMA

PROGRAMMING SUPPORT LIBRARY

TECHNICAL REPORT

MAY 1978

Submitted to:

Rome Air Development Center
Griffiss Air Force Base, New York

Under Contract F30602-77-C-0249

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
<i>NTIS</i>	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
<i>A</i>	<i>21</i>

Federal Systems Division
INTERNATIONAL BUSINESS MACHINES CORPORATION
Gaithersburg, Maryland

Table of Contents

<u>Section</u>	<u>Page</u>
1.0 INTRODUCTION	1
2.0 PSL CONVERSION	3
2.1 Conversion Schedule	3
2.2 Conversion Philosophy	3
2.2.1 Use of Program Production Library (PPL)	3
2.2.2 Structure Figures	3
2.2.3 INCLUDE	4
2.2.4 Top-Down Development	4
2.2.5 Design and Code Reviews	5
2.2.6 Programming Librarian	5
2.2.7 Chief Programmer Team	6
PHASE I	
3.0 PSL CAPABILITIES	7
3.1 System Organization	7
3.2 PSL Functions	12
3.2.1 Library Maintenance	13
3.2.1.1 INITIAL - Initialize a Project	13
3.2.1.2 CREATE - Create a Section	13
3.2.1.3 BACKUP	16
3.2.1.4 RESTORE	16
3.2.1.5 TERMINATE	16
3.2.2 Unit Maintenance	16
3.2.2.1 ADD - Add a Unit	17
3.2.2.2 REPLACE - Replace a Unit	17
3.2.2.3 CHANGE - Change a Unit	17
3.2.2.4 MOVE - Move a Unit	17
3.2.2.5 PURGE - Purge a Unit	17
3.2.3 Program Processing	17
3.2.3.1 COMPILE - Compile a Module	17
3.2.3.2 LINK - Link a Program	18
3.2.3.3 EXECUTE - Execute a Program	18
3.2.4 Output Processing	18
3.2.4.1 INDEX - Print a Section Index	18
3.2.4.2 SOURCE - Print a Card-Image Unit	20
3.2.4.3 MDPRINT - Print Reports	20
3.2.4.4 AUTHOR - Print by Author	25
3.2.4.5 DOCUMENT - Print Text	25
3.2.4.6 CSCAN - Print by Character String	25
3.2.5 General Functions	25
3.2.5.1 PARAM	25
3.2.5.2 JCL	25
3.2.6 Management Data Collection	29
3.2.6.1 MDPLAN - Management Data Plan	32
3.2.6.2 MDFORMAT - Management Data Format	
Maintenance	32
3.2.6.3 MDUPDATE - Manual Data Maintenance	32

Table of Contents (Continued)

<u>Section</u>		<u>Page</u>
3.2.6.4	MDXCHECK - Exception Checking	32
3.2.6.5	MDCOLLECT - Management Data Collection	32
3.3	General Capabilities	32
3.3.1	High-Level Parameters	32
3.3.2	Multi-Library Search	33
3.3.3	Independent Files	33
3.3.4	Spawned Jobs	34
3.3.5	Option Keyword	35
3.3.6	Stub Generation	35
3.3.7	Error Handling	36
3.3.8	INCLUDE Statements	36
3.3.9	Management Data Cycling	37
3.3.10	Management Data Archiving	37
3.3.11	Transportability	38
4.0	PHASE II PSL ENHANCEMENTS	
5.0	PSL SYSTEM TESTS	
5.1	DMAHTC System Test	
5.2	DMAAC System Test	
6.0	PSL DELIVERIES	

ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1	PSL System Flowchart	8
2	Hierarchical Structure of a Project	9
3	Organization of a PSL Section	11
4	Functions, Subfunctions and Keywords	14
5	Section Index Report	19
6	Structured COBOL (SCOBOL)	21
7	Program Structure Report (Part 1 of 2)	22
7	Program Structure Report (Part 2 of 2)	23
8	Management Data Report	24
9	AUTHOR Report	26
10	DOCUMENT Report	27
11	Character Scan Output	28
12	Management Data Base Structure	30
13	Management Report Structure	31

187

REFERENCES

- A. STRUCTURED PROGRAMMING SERIES, VOLUME I-XV, RADC-TR-74-300

Section 1

INTRODUCTION

Over the past several years the structured programming technology has gained wide use and acceptance throughout the Department of Defense. The concept of structured programming is 1) to define a program with a single entry point and a single exit point, and 2) to code in such a manner as to develop lower level functions as pieces which plug into higher level functions. By significantly reducing the number of interfaces, potential error points are eliminated. Structured programs tend to be less error prone, easier to read, easier to maintain; these characteristics obviously are attractive to cost-conscious projects with software contents.

The structured programming discipline requires tools to support the implementation. In 1974-1975 IBM's Federal Systems Division delivered to Rome Air Development Center (RADC) a 15-volume series, The Structured Programming Series, which 1) describes a set of structure figures which enable the implementation of structured code, and 2) defines tools and techniques which support the structured programming discipline. Those tools germane to this report include:

- a. precompilers which accept structured source code and translate the structure figures into unstructured code acceptable to the respective compiler;
- b. programming support libraries which maintain code at the unit (single page of code) level.

The series also includes guidelines on management data collection and reporting and documentation standards.

The structured programming discipline has further implications than coding a program using the defined structure figures and segmenting code into hierarchical units. Top-down, structured design is an obvious forerunner of structured programming. Integration and testing can also be accomplished in a top-down manner.

Structured source code can be made so readable that certain detailed documentation requirements can be waived.

The World-Wide Military Command-and-Control System originally

charged RADC with procuring a programming support library. RADC, in turn, contracted with IBM to develop a programming support library which would operate in the WWMCCS environment and provide the necessary capabilities to support the new programming technology as defined in the Structured Programming Series (Reference A).

As a follow-on to this effort, the DMA PSL conversion was begun in October 1977 and the DMA PSL system test was completed on May 11, 1978.

The DMA PSL is a comprehensive system software package which supports the growth and maintenance of structured programming projects in a top-down development environment. The system provides:

- a. A framework for the organization of a project
- b. Simple functional statements to interface between the programmer and the machine
- c. Structural and statistical reports for control of the development process and for communication between programmers.

The DMA PSL system provides special structured programming support for the Structured COBOL and Structured FORTRAN languages. However, unstructured programs may also be stored and maintained under the system.

The PSL system operates on a UNIVAC Series 1100 computer under an Executive System using a standard UNIVAC software and hardware configuration. Libraries are maintained on direct access storage devices. The system utilizes the standard card reader and printer and one tape drive.

The PSL system is designed to operate in the batch mode and is directed to perform specific functions by PSL Function cards.

Section 2

PSL CONVERSION

2.1 Conversion Schedule

The DMA PSL has been developed in two phases:

- a. Phase I converted WWMCCS PSL COBOL programs implemented prior to October 1977 to UNIVAC 1100 series ASCII COBOL to achieve an initial PSL system operation.
- b. Phase II converted new and enhanced capabilities incorporated in the WWMCCS PSL system subsequent to September 1977.

The original contract period covered seven months. Technical development was performed at DMAHTC via UNISCOPE terminals.

2.2 Conversion Philosophy

Not only does the PSL support structured code and top-down integration, the PSL itself is written in structured COBOL and was integrated in a top-down manner.

2.2.1 Use of Generalized Software Support

The UNIVAC 1100 series computer software was greatly relied upon to perform the conversion of PSL programs from the WWMCCS to the DMA environment. The Maryland Text Editor was essentially used to convert COBOL code to conform to UNIVAC 1100 series ASCII COBOL compiler standards and to modify and add source code as required. Full advantage was taken of the UNIVAC software facilities for maintaining program code and for compiling, loading and executing PSL programs, as frequently required in the conversion and testing activities.

2.2.2 Structure Figures

As part of the WWMCCS effort supporting the Structured Programming Series, a COBOL precompiler was specified and developed. This COBOL precompiler was made operational and used as a standalone program to compile structured program modules as they were subsequently converted.

The COBOL precompiler and all subsequent precompilers incorporated into the PSL support the following structure figures:

- a. CASEENTRY
 CASE
 ELSECASE
 ENDCASE
- b. DO WHILE
 ENDDO
- c. DO UNTIL
 ENDDO
- d. IF
 ELSE
 ENDIF

2.2.3 INCLUDE

True top-down integration is enabled by the INCLUDE statement. This directive indicates that the unit of code (approximately 50 lines) identified by the INCLUDE statement is to be compiled as if it were coded in-line. The source code of the unit containing the INCLUDE statement need not be altered to incorporate, at any later time, the INCLUDED unit of code.

2.2.4 Top-Down Conversion

Top-down conversion and integration was adhered to in the testing of each program module. The top-level program module is Batch-Control, BCTLE. As soon as BCTLE was converted, it was compiled and executed, thus beginning system integration immediately.

When BCTLE operation was such that other modules could be called, those modules were added to the system as they were converted. Each test on each new unit served as a regression test on higher-level units throughout the system hierarchy.

2.2.5 Design and Code Reviews

Another aspect of the new programming technology is the formal review, both design reviews and code reviews. The original PSL development team conducted both types of reviews, but frequently on an informal basis. Since the development team was relatively small, the informal review was satisfactory in many cases. A more formal review procedure was implemented during the development of the Management Data Collection and Reporting Subsystem.

The original PSL architecture and system design was the work of a single designer with informal review and critique by the department manager. As the development team increased in size and as many capabilities were being designed simultaneously, design sessions were held for early exchange of ideas and to take advantage of the "brainstorming" effect. Module design was then solidified into Program Design Language (PDL). Design reviews were conducted on the PDL. Copies were distributed to the reviewers two days before the scheduled review so that each participant would be prepared. The designer whose work was being reviewed conducted the review. For review of less complex functions, two reviewers participated. For review of for example, the Management Data Collection and Reporting Subsystem, all team members participated.

Code reviews were less formally conducted. In general, copies of the code were distributed to two other group members and written comments were returned to the programmer. The technical leader participated in almost all design and code reviews.

2.2.6 Programming Librarian

The PSL project utilized the services of a programming librarian throughout program development.

The librarian was responsible for maintaining both the internal and external libraries. Each programmer had one or more development libraries. The librarian maintained these libraries as directed by the programmer, adding new code and changing or deleting existing code. Daily an Index Report was obtained for each library. Whenever an external library was out of synchronization with an internal library, the appropriate listings were generated and filed. Theoretically, the internal and external libraries should be under librarian control and remain synchronuous. However, programmers frequently worked

variable hours to accommodate computer availability. When such a situation occurs, a single librarian is not able to maintain complete control and must rely on such techniques as described.

The integration library was maintained under the direction of the technical leader. Programmers were not permitted to update the integration library but informed the technical leader when code was sufficiently tested and ready to be added to the integration library. The internal and external library formats and the librarian's duties were similar to those for the development libraries.

2.2.7 Chief Programmer Team

The PSL project did not use a Chief Programmer as such. Instead, the technical leader was the lead designer, programmed a large percentage of the most complex functions, had technical cognizance over the complete PSL, trained less experienced programmers, and participated in design and code reviews. The major difference was that significant portions of design and coding were performed by other team members.

Section 3

PHASE I PSL CAPABILITIES

The PSL system can be used to support a varied set of programmer needs, ranging from a small program to large and complex systems. The user has an extensive selection of options, for which default values are provided where practical. Thus a programmer may begin to use the system in a simple straightforward manner and gradually enlist the aid of the more specialized services of the system. A general flowchart is shown in Figure 1.

3.1 System Organization

The PSL system is designed to support the program development and maintenance effort of an entire organization. In a large organization, the system must support many persons working on different programming projects which may be completely unrelated. The PSL provides means of maintaining control over all the data related to each project.

One means of control is the convention used for identifying and organizing the data (source code, compiled modules, test data, etc.) stored under a project. This convention subdivides the data, beginning with the programming project level and proceeding down to single logical units of data. The hierarchical structure of a project is shown in Figure 2.

There are four levels of data identification under the PSL. The highest level of identification is a project. This corresponds to a major programming operation and consists of all of the data related to that operation. Under the DMA PSL system, a project is equivalent to a file name qualifier.

The next level of identification is a library. Each project is composed of one or more libraries. Multiple libraries can be used effectively to provide version control over the programming process and to support top-down program development and integration. Any number of libraries may exist under a given project.

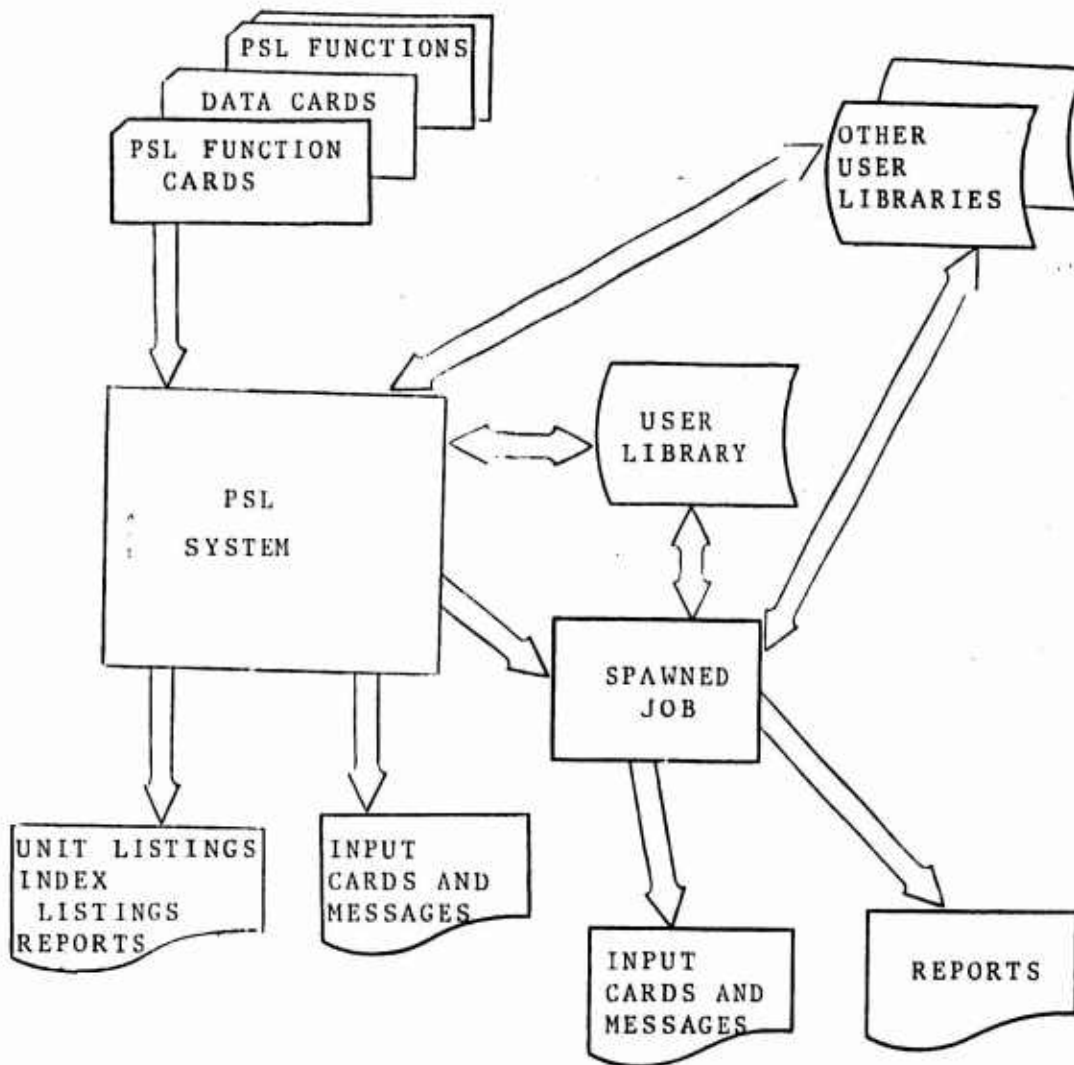


Figure -1 . PSL System Flowchart

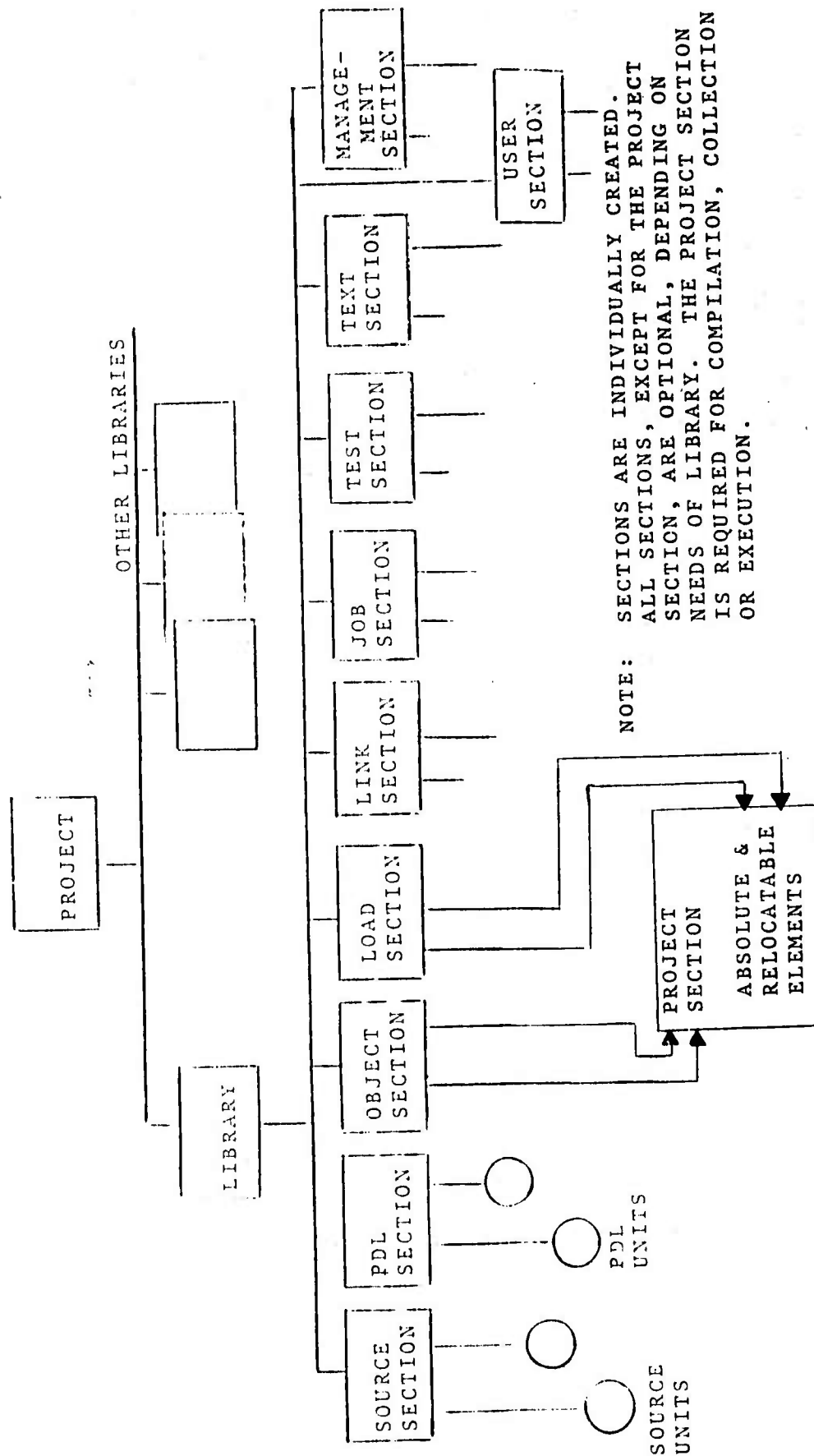


Figure-2 Hierarchical Structure of a Project

A library is composed of segments of programming data grouped according to type of data. Each type of data is stored in a specific section of a library. The names of a section is one of the following predefined section names:

- a. SOURCE - source code statements
- b. OBJECT - object modules from compilation
- c. LOAD - load modules for execution
- d. LINK - loader control cards
- e. JOB - job control cards used during execution of user program
- f. TEST - test data for use by user program
- g. PDL - Program Design Language statements
- h. TEXT - documentation
- i. MGMT - management data

The PSL system manipulates the data for each section appropriately for that type of section.

A section in a library under the PSL system is created in the format of a PSL Standard File.

The organization of a section is shown in Figure 3. The PSL Standard File is a COBOL random (relative access) file. The file contains 128-word blocks (768 6-bit characters). A Standard PSL File will be created for each section when the CREATE function is invoked for the section. The actual name of a section file is a concatenation of a one-character section code and the name of the user's library. Thus each section within each library, under a given project, is represented by a PSL Standard File, which is catalogued with the qualifying project ID.

The first block in a PSL Standard File is the first Control Block for the file. It contains information pertinent to the entire section, such as section name, user-selected options, file size, and the block flags which are used to control the space allotment within the file. The first Control Block also contains the block number of the first Index Block for the file.

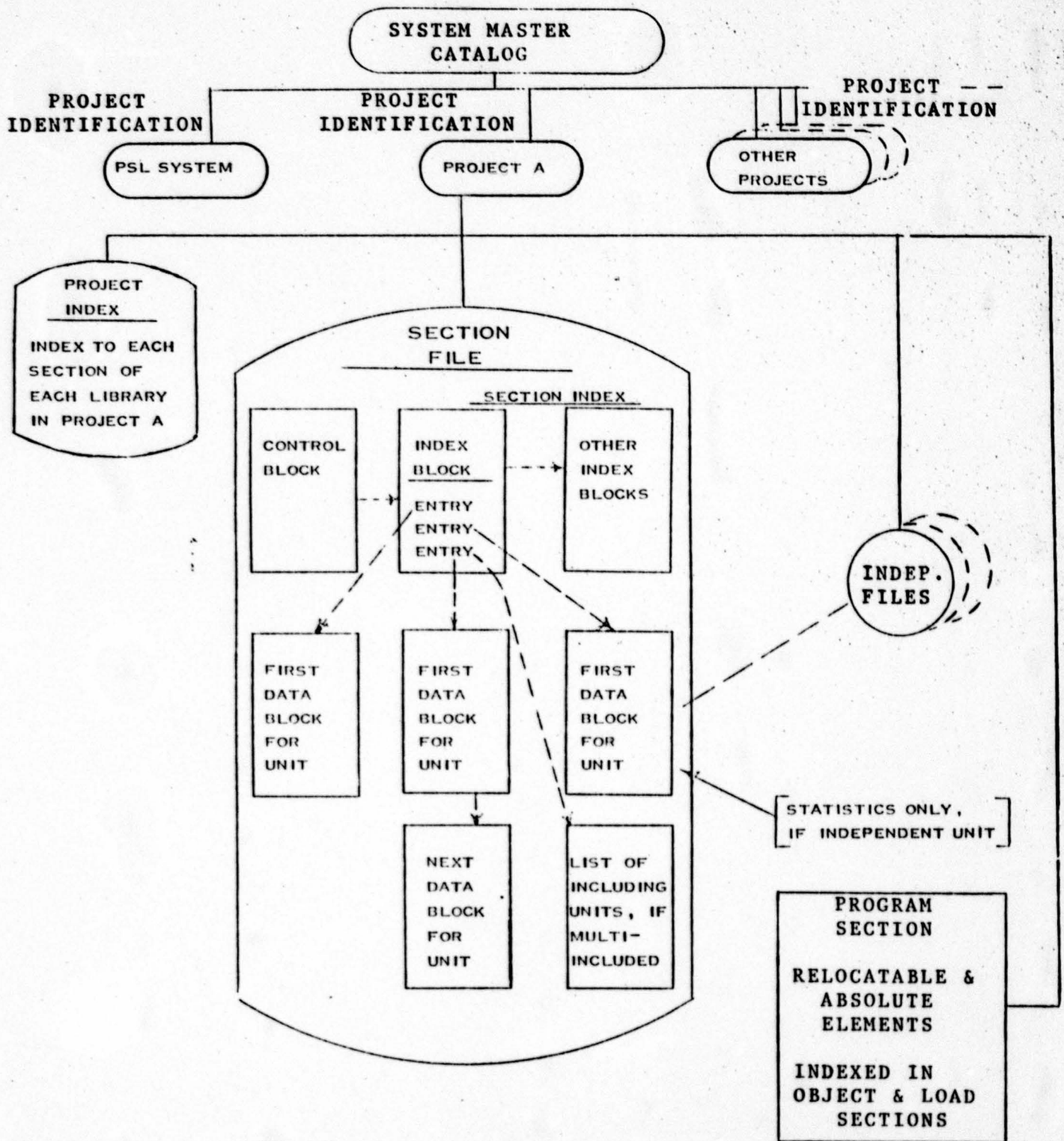


Figure-3

Organization of a PSL Section

The Index Blocks contain entries pointing to the location of each data unit within the section. Initially, only one block is assigned to the index. When that block is full, another Index Block is developed and half of the entries from the first block are moved to the new block.

A data unit within a section may be stored in blocks within the PSL Standard File or in a completely separate sequential file, referred to as an independent file. The section type and unit type determine which format is used for any given unit. Structured SOURCE and PDL units and straightforward card-image data are usually stored in blocks within the PSL Standard File. OBJECT and LOAD units are stored in a program file which is created for the particular library to which they belong.

A section is composed of logical segments of data called units. The unit is the lowest level of the naming convention used under the PSL system. A unit is therefore uniquely identified by the following set:

- a. Project name
- b. Library name
- c. Section name
- d. Unit name

The actual structure and content of a unit is related to the type of section to which the unit belongs.

3.2 PSL Functions

PSL Functions are grouped into six categories:

- a. Library Maintenance
- b. Unit Maintenance
- c. Program Processing
- d. Output Processing
- e. General Functions
- f. Management Data Collection

The input to the PSL system consists of directives on input cards (PSL Function cards), data from the user's libraries, and, in some instances, an input tape. The PSL system accepts PSL Function cards as batch-input data cards. The user's own program and data cards are interspersed, as necessary, with the PSL Function cards. The characters "*** "

in columns 1 through 3 identify a card as a PSL Function card or a Function continuation card. The PSL cards contain the name of a Function requested and, if appropriate, sets of keywords and value-entries. Figure 4 lists the PSL Functions, and associated keywords. The functions under each of the six categories are described in the following paragraphs.

3.2.1 Library Maintenance

Before programs are developed under the PSL system, a user establishes his project and library. The following PSL Functions are used for general library maintenance.

3.2.1.1 INITIAL - Initialize a Project

Under the PSL system, a user stores programs and data in discrete units, within sections of a library, under a project. A project must be initialized before libraries are built under it. The PSL Function INITIAL establishes the project as a PSL Project and prepares an index for library-sections under the project.

3.2.1.2 CREATE - Create a Section

After a project is initialized, the CREATE Function is used to build sections in the user's library. The library name uniquely identifies a group of standard sections under a project.

Within a library, the PSL system allows the user to create any of the eight standard PSL sections:

- a. SOURCE - This section contains all of the source code which is stored in a library, regardless of language or structure. Unit of structured code, which at present include Structured COBOL (SCOBOL) and Structured FORTRAN (SPFORT) are stored in blocks in a random file. It is recommended that the size of structured units be restricted to one page of code. For compilation, structured units are combined in accordance with the INCLUDE statements found in the units. Units of unstructured code (ASM, FORTRAN, and COBOL) are stored as separately compilable units in sequential files.

Functions

ADD
AUTHOR
BACKUP
CHANGE
COMPILE
CREATE
CSCAN
DOCUMENT
EXECUTE
INDEX
INITIAL
JCL
LINK
MDCOLLECT
MDFORMAT
MDPLAN
MDPRINT
MDUPDATE
MDXCHECK
MOVE
PARAM
PURGE
REPLACE
RESTORE
SOURCE
TERMINATE

Subfunctions

COPY
DELETE
EJECT
HEADER
INSERT
MODIFY
SHIFT
SPACE
TEXT

KeywordsPSL

AFTER	OP
ALL	OPGMR
ARCHIVE	OPTION
BACKUP	PAGE
CALL	
CLASS	PASSWORD
COLUMN	PGMR
COMPRESS	PLINES
CYCLE	PROCEDURE
ELEMENT	PROJ1
FMS	.
FROM	.
HISTORY	.
HPRINT	PROJ9
HSPACE	PROJECT
INDENT	RECYCLE
JOB	REPLACE
KEY	REPORT
LADJUST	SECTION
LANGUAGE	SPCHECK
LEVEL	SPLength
LIB1	STANDARD
.	START
.	STRING
.	SYSTEM
LIB9	SUBSYSTEM
LIBRARY	TO
LINE(S)	UNIT
LINK	
LOAD	UPGMR
	USPACE
LSPACE	UTYPE
MEDIA	
MGMTDATA	
MOD	
MODULE	
NBRLINES	
NEST	
OBJECT	
OLDLIB	
OLDPROJ	
OLDSEC	
OLDUNIT	

Figure 4. Functions, Subfunctions, and Keywords

- b. OBJECT - The object modules (i.e., relocatable elements) which result from compilation of source code are stored in the OBJECT section. The name of the object unit is the same as the name of the top-most unit of source code. Object modules are used singly or in combination to form a load for execution.
- c. LOAD - If the load module is to be saved on permanent storage, it is stored as a system-loadable program in the LOAD section. Units in the OBJECT and LOAD sections are produced by UNIVAC system functions and are therefore not maintained with unit maintenance Functions, as are the other sections. A load module (i.e., absolute element) is produced by Collector, using one or more object modules.
- d. LINK - Units in this section are used to store control cards for directing the mapping process when more than one object module is to be loaded.
- e. JOB - Control cards may be stored in units in the JOB section and invoked by the EXECUTE Function. The cards will be added to the program execution activity to provide the job control cards the user's program requires.
- f. TEST - This is a general section for card-format data usually used for test input.
- g. PDL - Program Design Language statements are stored in units in the PDL section. PDL consists of English-like statements which follow the basic rules of structured programming and are used to define the program structure and logic.
- h. TEXT - This section contains units of standard textual material, which are written primarily for use as program documentation. The units may be maintained under the PSL system and printed as any other card-format unit.

- i. MGMT - A management plan is initialized in the created MGMT section and subsequently updated to define the project elements that comprise the management data report requirement. Format units are established to define the report contents and input units are added to introduce manual data. Both automatic data from unit accounting records and manually input data are collected and archived in collection units for subsequent input to management data reports.

3.2.1.3 BACKUP

The PSL system provides a BACKUP Function to save the user's programs and data on tape. The BACKUP may be invoked for a complete project, a library, or a section.

3.2.1.4 RESTORE

The RESTORE Function is used to write the contents of the BACKUP tape into the library. This Function may be used to restore the complete contents of the BACKUP tape, or it may selectively recover a portion of the total data tape at as low a level as a unit of code.

3.2.1.5 TERMINATE

This Function enables a user to delete a section, a library, or an entire project. Files which are released are overwritten.

3.2.2 Unit Maintenance

After a user has initialized a project and has created the sections which are needed in his library, data may be stored in these units. The user may add and update data in those sections which are used for card-image data (SOURCE, PDL, LINK, JOB, TEST, and TEXT). Units in the OBJECT and LOAD sections are not maintained directly by the user, since they are automatically created and updated by the PSL system as a result of the COMPILE and LINK Functions (see subsection 3.1.3). MGMT units are updated by the management data collection Functions.

The PSL system automatically maintains accounting information for each unit in each section. This information is initialized when the unit is created and updated when the unit is modified.

The following PSL Functions are used to add, change, and purge units of card-image data in sections (other than MGMT) in the user's library.

3.2.2.1 ADD - Add a Unit

The ADD Function is used for the original introduction of data into a unit. The accounting information for the unit is initialized by the ADD Function. The actual data cards for the new unit follow the ADD Function card in the run stream. After the ADD is completed, a listing of the unit will be automatically generated.

3.2.2.2 REPLACE - Replace a Unit

After data has been added to a unit, the REPLACE Function is used to completely replace all of the data lines in the unit. Accounting information is not re-initialized, but is updated.

3.2.2.3 CHANGE - Change a Unit

The CHANGE Function is used to modify the contents of a unit. Modification details are provided on Subfunction cards (COPY, DELETE, INSERT, MODIFY, and SHIFT) which follow the CHANGE card. New source statements follow the INSERT and MODIFY Subfunction cards. After the CHANGE operation is completed, a listing of the altered unit will be automatically generated.

3.2.2.4 MOVE - Move a Unit

Units are moved from one library to another, or within a library, with the MOVE Function.

3.2.2.5 PURGE - Purge a Unit

An individual unit is removed from a library with the PURGE Function. If the PURGE of a unit results in the release of a file, the file contents will be overwritten.

3.2.3 Program Processing

The PSL system provides many facilities to support the compilation, loading, and execution of programs. One of the more powerful capabilities available for these Functions is the optional search through multiple libraries during retrieval.

3.2.3.1 COMPILE - Compile a Module

The COMPILE Function retrieves units from the SOURCE section, invokes the appropriate precompiler for structured source code, compiles (or assembles) the resulting stream of code, and stores the compiler product in a unit in the OBJECT section. The UNIT name is the name of the top-most source unit of the

module which is to be compiled. This name will be used for the name of the compiled OBJECT unit. The processing which is invoked by the COMPILE Function depends upon the language of the unit. Under the present PSL system, procedures exist to process languages COBOL, SCOBOL, (Structured COBOL), FORTRAN, SPFORT (Structured FORTRAN) ASM.

3.2.3.2 LINK - Link a Program

The LINK Function retrieves one or more object modules from the OBJECT section, loads the object modules, and retains a permanent copy of the system-loadable file. The resulting load module is stored as a unit in the LOAD section.

If more than one object module is to be linked together to form the load module, COLLECTOR control cards are retrieved from a unit in the LINK section.

3.2.3.3 EXECUTE - Execute a Program

Job control cards for execution of a program are previously stored as a unit in the JOB section. The EXECUTE Function invokes the execution of a program using the indicated job control cards as the input run stream.

The program itself may be retrieved by the PSL system in one of the following forms:

- a. A load module from the LOAD section
- b. An object module from the OBJECT section which will be processed by the COLLECTOR.
- c. A series of object modules which are selected as a result of COLLECTOR control cards in a unit of the LINK section and which will be processed by the COLLECTOR.

3.2.4 Output Processing

Six PSL Functions are available in the PSL system to obtain reports.

3.2.4.1 INDEX - Print a Section Index

The INDEX Function prints a status report for a section. The report contains information pertaining to the section as a whole, as well as a listing of the index for the section. An example of a Section Index Report is shown in Figure 5.

UNIT	INCLUDED	VER/MOD	UPDATED	UPDATED	LANGUAGE	LINES	CREATED
*****	TYPE	COUNT	*****	BY	*****	*****	*****
PROCESS-FUNCTION	STUB	1	000/000	10/07/76	12109	PSLTEST	SCOBOL 0 10/07/76
SPAWN-A-JOB	STUB	1	000/000	10/07/76	12109	PSLTEST	SCOBOL 0 10/07/76
TIPTOP	MAIN		001/000	10/07/76	12109	PSLTEST	SCOBOL 14 10/07/76
TIPTOP-ENVIRONMENT-DIVISION	INCLUDED	1	001/000	10/07/76	12109	PSLTEST	SCOBOL 25 10/07/76
TIPTOP-FILE-SECTION	INCLUDED	1	001/000	10/07/76	12109	PSLTEST	SCOBOL 10 10/07/76
TIPTOP-PROCEDURE-DIVISION	INCLUDED	1	001/000	10/07/76	12109	PSLTEST	SCOBOL 41 10/07/76
TIPTOP-WORKING-STORAGE	INCLUDED	1	001/000	10/07/76	12109	PSLTEST	SCOBOL 42 10/07/76

Figure 3 Section Index Report

3.2.4.2 SOURCE - Print a Card-Image Unit

The SOURCE Function can be used to print a listing of any unit which is in card-image format. This includes the units in the SOURCE, PDL, LINK, JOB, MGMT, TEST, and TEXT sections. If the unit is from the SOURCE or the PDL section and the unit is written in a supported structured language (Structured COBOL, and Structured FORTRAN), the structured code is automatically indented for the proper alignment of the structures on the listing. This listing is always provided when a unit is added to the library or modified. The contents of the unit may, on option, be written to tape or punched on cards. In these latter cases, the header information is omitted and the code is reproduced as it exists in the unit, without automatic indentation. An example of a listing of structured COBOL (SCOBOL) is shown in Figure 6.

3.2.4.3 MDPRINT - Print Reports

The MDPRINT Function is used to print management data reports. There are two categories of such reports:

- a. Program Structure Report
- b. Management Data Report

The Program Structure (PS) report begins with the unit which is named on the Function card and develops a hierarchically nested and indented list of all units which are referenced by an INCLUDE statement, either in the originally-requested unit or in units which are themselves INCLUDED in the hierarchical program structure. Units which are referenced by a CALL statement are also printed with the appropriate indentation in the list, but they are not automatically searched for lower levels of INCLUDE or CALL statements. An option is available to invoke a PS report for all CALLED units. Statistical organization information is printed for each unit. An alphabetically-arranged list of all referenced units follows the hierarchical list. An example of a Program Structure Report is shown in Figure 7.

A Management Data report prints the contents of one or more management data units. Management data units may contain a combination of automatically collected data and manually input data combined according to the specifications in an associated user-defined format unit. A general format is provided to accommodate a wide variety of data collections to be printed. A representative Management Data Report is shown in Figure 8.

```

1  BATCH-PSL-CONTROL.
2  DISPLAY " TRACE - BCTL (BATCH-PSL-CONTROL) EXECUTED.".
3  MOVE SPACES TO PARAMETER-TABLE.
4  CALL OBUSE
5      USING USERID-FROM-CARD. IDENT-INFO.
6  MOVE USERID-FROM-CARD TO
7      USERID OF PARAMETER-TABLE.
8  PROGRAMMER-NAME OF PARAMETER-TABLE.
9  OPEN INPUT INPUT-CARDS
10     OUTPUT MESSAGE-FILE.
11  MOVE "ADD" TO SHORT-INPUT-FUNCTION
12  READ INPUT-CARDS
13     AT END
14     MOVE ZERO TO PROCESSING-STATUS.
15  IF PROCESSING-STATUS NOT EQUAL TO CODE-FOR-END-INPUT-CARDS
16     CALL OBFN
17     USING INPUT-CARD-FUNCTION. PROCESSING-STATUS.
18  DO UNTIL NORMAL-STATUS
19     MOVE HIGH-FUNCTION-VALUE TO FUNCTION-NUMBER.
20     SEARCH ALL PSL-FUNCTIONS
21     WHEN PSL-FUNCTION (PF-INDEX) EQUAL TO
22         SHORT-INPUT-FUNCTION
23         SET FUNCTION-NUMBER TO PF-INDEX.
24     INCLUDE PROCESS-FUNCTION.
25     CALL OBFN
26     USING INPUT-CARD-FUNCTION. PROCESSING-STATUS.
27  ENDDO
28  MOVE CODE-FOR-END-INPUT-CARDS TO MESSAGE-NUMBER.
29  CALL PRMS
30     USING MESSAGE-NUMBER. MESSAGE-DATA.
31  IF SPAWN-JOB
32     INCLUDE SPAWN-A-JOB.
33  ENDIF
34  ELSE
35     MOVE CODE-FOR-NO-INPUT-CARDS TO MESSAGE-NUMBER.
36     CALL PRMS
37     USING MESSAGE-NUMBER. MESSAGE-DATA.
38  ENDIF
39  CALL RLAF
40  CLOSE INPUT-CARDS. MESSAGE-FILE.
41  STOP RUN.

```

Figure-6 STRUCTURED COBOL (SCOBOL)

UNIT LEVEL	UNIT LINES	UNIT NAME	UNIT TYPE	ORIGINATE DATE	LAST UPDATE	PROJECT NAME	LIBRARY NAME	SP FLAG
1	14	TIPTOP	MAIN	07 OCT 76	07 OCT 76	FHACD129	NEWCODE	
2	23	TIPTOP-ENVIRONMENT-DIVISION	REAL-SINGLE-INCL	07 OCT 76	07 OCT 76	*	*	
2	16	TIPTOP-FILE-SECTION	REAL-SINGLE-INCL	07 OCT 76	07 OCT 76	*	*	
2	42	TIPTOP-WORKING-STORAGE	REAL-SINGLE-INCL	07 OCT 76	07 OCT 76	*	*	
2	41	TIPTOP-PROCEDURE-DIVISION	REAL-SINGLE-INCL	07 OCT 76	07 OCT 76	*	*	
3		OBUSE	CALLED					
3		OBFN	CALLED					
3		PROCESS-FUNCTION	STUB-SINGLE-INCL	07 OCT 76	07 OCT 76	FHACD129	NEWCODE	
3		OBFN	CALLED					
3		PRMS	CALLED					
3		SPAWN-A-JOB	STUB-SINGLE-INCL	07 OCT 76	07 OCT 76	FHACD129	NEWCODE	
3		PRMS	CALLED					
3		RLAF	CALLED					

SP-FLAG-MESSAGES (NBRS, UNDER SP-FLAG HEADER ARE CUMULATIVE) * - SAME NAME

1 - UNIT CONTAINS MOD TO- STATEMENTS

2 - NBR. OF LINES EXCEEDS DEFINED LIMITS

4 - MORE THAN ONE STATEMENT ON A LINE

Figure-7 Program Structure Report
(Part 1 of 2)

CROSS REFERENCE LISTING

UNIT NAME	PROJECT NAME	LIBRARY NAME	UNIT TYPE	HIGHER UNIT NAME
OBFN			CALLLED	TIPTOP-PROCEDURE-DIVISION
OBUSE			CALLLED	TIPTOP-PROCEDURE-DIVISION
PRMS			CALLLED	TIPTOP-PROCEDURE-DIVISION
PROCESS-FUNCTION		NEWCODE	STUB-SINGLE-INCL	TIPTOP-PROCEDURE-DIVISION
RLAF			CALLLED	TIPTOP-PROCEDURE-DIVISION
SPAWN-A-JOB		NEWCODE	STUB-SINGLE-INCL	TIPTOP-PROCEDURE-DIVISION
TIPTOP		*	MAIN	TOP OF TREE
TIPTOP-ENVIRONMENT-DIVISION		*	REAL-SINGLE-INCL	TIPTOP
TIPTOP-FILE-SECTION		*	REAL-SINGLE-INCL	TIPTOP
TIPTOP-PROCEDURE-DIVISION		*	REAL-SINGLE-INCL	TIPTOP
TIPTOP-WORKING-STORAGE		*	REAL-SINGLE-INCL	TIPTOP
	FHACD129			
	FHACD129			
	*			
	*			
	*			
	*			
	*			

PROJECT TITLE	NEWCODE-TITLE	NEWCODE PROJECT DESCRIPTION LINE 1	PROJ-TITLE	
PROJECT DESCRIPTION			PROJ-DESCR	010
PROJECT START DATE	770501		START-DATE	020
ESTIMATED COMPLETION DATE	770901		EST-END-DATE	030
ACTUAL COMPLETION DATE	0		ACT-END-DATE	040
PLANNED AVERAGE YEARS EXPERIENCE MANAGERS	10		P-AVE-MGRS	050
PLANNED AVERAGE YEARS EXPERIENCE ANALYSTS	12		P-AVE-ANAL	060
PLANNED AVERAGE YEARS EXPERIENCE PROGRAMMER	5		P-AVE-PROG	070
PLANNED AVERAGE YEARS EXPERIENCE ADMINISTRATIVE	2		P-AVE-ADMIN	080
PLANNED AVERAGE YEARS EXPERIENCE OTHER	0		P-AVE-OTH	090
ACTUAL AVERAGE YEARS EXPERIENCE MANAGERS	8		A-AVE-MGRS	100
ACTUAL AVERAGE YEARS EXPERIENCE ANALYSTS	11		A-AVE-ANAL	110
ACTUAL AVERAGE YEARS EXPERIENCE PROGRAMMERS	7		A-AVE-PROG	120
ACTUAL AVERAGE YEARS EXPERIENCE ADMINISTRATIVE	3		A-AVE-ADMIN	130
ACTUAL AVERAGE YEARS EXPERIENCE OTHER	1		A-AVE-OTH	140
PLANNED NUMBER OF MANAGERS	4		P-NBR-MGRS	150
PLANNED NUMBER OF ANALYSTS	5		P-NBR-ANAL	160
PLANNED NUMBER OF PROGRAMMERS	16		P-NBR-PROG	170
PLANNED NUMBER OF ADMINISTRATIVE	4		P-NBR-ADMIN	180
PLANNED NUMBER OF OTHER	2		P-NBR-OTH	190
ACTUAL NUMBER OF MANAGERS	4		A-NBR-MGRS	200
ACTUAL NUMBER OF ANALYSTS	4		A-NBR-ANAL	210
ACTUAL NUMBER OF PROGRAMMERS	12		A-NBR-PROG	220
ACTUAL NUMBER OF ADMINISTRATIVE	2		A-NBR-ADMIN	230
ACTUAL NUMBER OF OTHER	1		A-NBR-OTH	240
ESTIMATED PERSONNEL TURNOVER RATE	4		E-TURNOVER	250
ACTUAL PERSONNEL TURNOVER RATE	0		A-TURNOVER	260
ESTIMATED LOCAL TRAVEL CHANGE	30		E-LOC-TRIPS	270
ACTUAL LOCAL TRAVEL	60		A-LOC-TRIPS	280
ESTIMATED DISTANT TRAVEL	6		E-DIS-TRIPS	290
ACTUAL DISTANT TRAVEL	0		A-DIS-TRIPS	300
ESTIMATED WORKING CONDITIONS	5		E-WORK-COND	310
ACTUAL WORKING CONDITIONS	6		A-WORK-COND	320
PLANNED PROGRAMMING LANGUAGE EXPERIENCE	3		P-LANG-EXP	330
ACTUAL PROGRAMMING LANGUAGE EXPERIENCE	5		A-LANG-EXP	340
PLANNED SIMILAR APPLICATION EXPERIENCE	3		P-SIM-EXP	350
ACTUAL SIMILAR APPLICATION EXPERIENCE	6		A-SIM-EXP	360
PLANNED TARGET COMPUTER EXPERIENCE	3		P-TARG-EXP	370
ACTUAL TARGET COMPUTER EXPERIENCE	5		A-TARG-EXP	380
ESTIMATED CUSTOMER APPLICATION EXPERIENCE	3		E-APPL-EXP	390
ACTUAL CUSTOMER APPLICATION EXPERIENCE	4		A-APPL-EXP	400
ESTIMATED CUSTOMER EQUIPMENT EXPERIENCE	2		E-EQUIP-EXP	410
ACTUAL CUSTOMER EQUIPMENT EXPERIENCE	1		A-EQUIP-EXP	420
				430

3.2.4.4 AUTHOR - Print by Author

The AUTHOR Function can be used to print a list of unit names, or listings of the actual units, which were originally generated by and/or updated by a specific programmer. An example of an AUTHOR Report is shown in Figure 9.

3.2.4.5 DOCUMENT - Print Text

The DOCUMENT Function is used to print documentation stored in a library in the form of program design language, structured source code, text, etc. Output requirements are specified through keyword options provided on subfunction cards (HEADER, TEXT, EJECT, and SPACE). An example of a DOCUMENT Report is shown in Figure 10.

3.2.4.6 CSCAN - Print by Character String

The CSCAN Function is used to scan all units of the indicated section to locate a specific character string. The string may be up to 48 characters in length. The resulting output will be a list of the unit names containing the specific character string and the corresponding lines of code for each occurrence. Figure 11 contains an example of character scan output.

3.2.5 General Functions

Two PSL Functions are available which have general application in conjunction with other Functions.

3.2.5.1 PARAM

This Function is used to enter high-level parameters which apply to a series of subsequent Functions. The particular parameters which may appear on a PARAM card (PROJECT, LIBRARY, SECTION, PASSWORD, CLASS and PGMR) are cumulative parameters. When any of these parameters is established, it remains in effect. A PARAM Function card is ordinarily used to establish these values, but they may also be established by Function cards for which the keywords are valid.

3.2.5.2 JCL

The JCL Function enables the user to introduce JCL cards into the input stream of a subsequently spawned job. The Function is not required in the basic use of the PSL system, but provides the flexibility of using additional GCOS control cards in conjunction with such PSL Functions as COMPILE and LINK.


```

*      THE MODULE TIPTOP PROCESSES THE ADD A UNIT FUNCTION (ADUN)
TIPTOP
  INITIALIZE PARAMETER-TABLE WITH SPACES
  CALL OBUS
  MOVE PGMR-NAME TO PARAMETER-TABLE
  OPEN INPUT INPUT-CARDS.
      OUTPUT MESSAGE-FILE
  MOVE "ADD" TO INPUT-FUNCTION
  READ INPUT-CARDS
  IF NOT END OF INPUT CARDS
      CALL OBFN.
      DO UNTIL NORMAL RETURN FROM OBFN
          SEARCH PSL-FUNCTIONS
          WHEN PSL-FUNCTION IN THE TABLE EQUAL INPUT-FUNCTION
              SET FUNCTION-NUMBER TO INDEX OF TABLE.
          INCLUDE PROCESS-FUNCTION.
          CALL OBFN.
      ENDDO.
      CALL PRINT MESSAGE (END OF INPUT CARDS).
      IF SPAWN JOB NEEDED
          INCLUDE SPAWN-A-JOB
      ENDIF.
  ELSE
      CALL PRINT MESSAGE (NO INPUT CARDS)
  ENDIF.
  CALL RLAF.
  CLOSE INPUT-CARDS.
      MESSAGE-FILE.
  STOP RUN.

```


3.2.6 Management Data Collection

The Management Data Collection and Reporting (MDCR) capability enables software projects utilizing the PSL facility to obtain statistical data reports on the status and progress of project activities. Figure 12 illustrates a typical MDCR Data Base composed of a single Management (MGMT) section library and multiple Source section libraries. The set of top units and included units present in these libraries constitutes the total code under development for a given project. Statistics relating to unit update activity are maintained in the individual Unit Accounting records. These "automatic" statistics may be collected and summarized through Management Data Functions.

A management plan is defined to describe the hierarchical organization of data reports whose generation is illustrated in Figure 13. Format units are established for each of the reported data levels described in the management data plan to determine the order and source of data items to be collected and summarized.

Module level data may be derived from essentially two data sources; that is, 1) unit accounting records and 2) manual inputs. Job level data is manually input only. Manual data can be added, changed, or deleted.

Manual inputs are also made to maintain exception check specifications. Any two numeric data items at the same format level may be subjected to a value "variance" check. The data item values are compared at the time the data is collected and a determination made as to whether an exception exists. If an exception does exist, pertinent information is added to the data collection unit so that management reports which are subsequently produced from that collection may flag the excepted data item.

A data collection is performed according to an indicated management data plan. The ensuing collection activity automatically verifies and cross-relates plan level elements, format requirements and data source availability. Data collection will proceed until the entire management plan is processed and the collected data is stored in the designated MGMT section.

After data has been collected, a management report can be generated upon request. The management data report has a general format which provides appropriate identification of the level of data and the specific items for which values are printed. Figure 8 contains a sample management data report.

Applicable PSL Functions	* * * * * PSL MGMT Section * * * * *					
	Management Data Plan Unit (MDCR-PLAN)					
	SYSTEM = BIGTOP					
	JOB = BIGJOB					
	SUBSYS=SUB1, MODULE=MOD1A, MODULE=MOD1B					
	SUBSYS=SUB2, MODULE=MOD2A, MODULE=MOD2B					
MDFORMAT	Management Data Format Units (MDCR-FORMAT-)					
	SYSTEM	SUBSYS	MODULE	JOB	UNIT	
MDUPDATE	Management Data Input Units					
	BIGTOP	BIGJOB	SUB1	SUB2	MODIA	ETC.
MDXCHECK						
MDCOLLECT PURGE	Management Data Collection Units					
	MDCR-COLLECTION		MDCR-ARCHIVE-001-(DATE1)			
			MDCR-ARCHIVE-002-(DATE2)			
			MDCR-ARCHIVE-003-(DATE3)			
* * * * * PSL SOURCE Section(s) * * * * *						
ADD CHANGE MOVE REPLACE PURGE	Top Units		Included Units			
	MOD 1A					
	MOD 1B					
	MOD 2A					
	MOD 2B					

Figure 12. Management Data Base Structure

Management Plan Input

vs. Management Reports Output

System = PSL-project
Job = PSL-system-test
Subsystem = PSL-main
Module = BCTLE
Module = PPLKE

.
. .
. .
. .
. .

Subsystem = PSL-Function
Module = ADUNE
Module = CHUNE
Module = CRFLE

.
. .
. .
. .
. .
. .

Subsystem = PSL-auxiliary
Module = ADXEE
Module = CHXXE

.
. .
. .

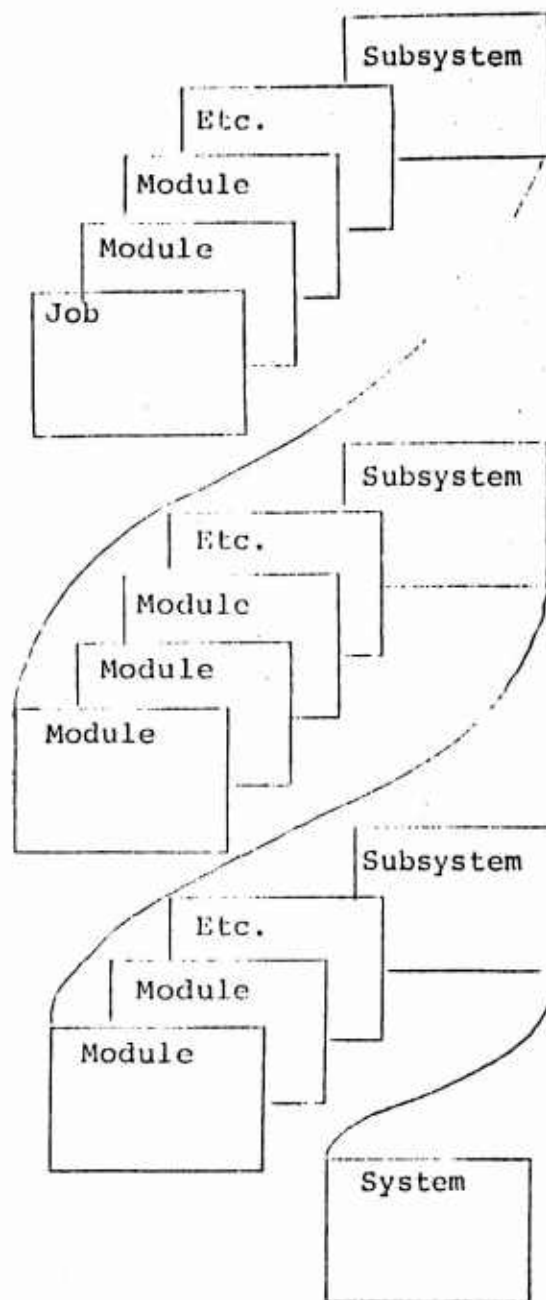


Figure 13. Management Report Structure

3.2.6.1 MDPLAN - Management Data Plan

The Management Data Plan Function is used to define the management data report structure.

3.2.6.2 MDFORMAT - Management Data Format Maintenance

The Management Data Format Function is used to define the data items that may be reported. Both automatically collected data items and manually input data items may be defined. Format units can be added, changed, or deleted by MDFORMAT as required.

3.2.6.3 MDUPDATE - Manual Data Maintenance

The Management Data Update Function is used to maintain the manually provided data which resides in management data units. These data can be added, changed, or deleted as required.

3.2.6.4 MDXCHECK - Exception Checking

The Management Data Exception Checking Function is used to annotate "excepted" data values in a management report. Exception check specifications are maintained in a management data unit corresponding to a denoted element in the report structure (i.e., plan unit). The specifications reference numeric data items contained in the associated record level format.

3.2.6.5 MDCOLLECT - Management Data Collection

The Management Data Collection Function is used to collect and archive the data designated by MDPLAN, MDFORMAT, MDUPDATE, and MDXCHECK. Recycling of cyclic data accumulations and archiving data are options of the MDCOLLECT Function.

3.3 General Capabilities

The following subject have general application over several PSL Functions. These general capabilities greatly enhance the serviceability of the PSL.

3.3.1 High-Level Parameters

The following parameters are cumulative during execution of the PSL system:

- a. PROJECT Project name
- b. LIBRARY Library name
- c. SECTION Section name

- d. PASSWORD Section password
- e. CLASS Security classification code
- f. PGMR Programmer name

Once the value for each of these keywords is established, either by a Function card or by default, it remains in effect. All of these parameters, except CLASS, may be replaced by another value by using the keyword on another Function card. A PARAM Function card is ordinarily used to establish one or more of these values, but the keywords may appear on any Function card for which they are valid. Use of a PARAM Function card eliminates the need for repetition of the same keywords on all subsequent Function cards utilizing those keywords.

Note on passwords:

A section password is required to access a section if the password was assigned when the section was created and if the processing will modify the contents of the section.

3.3.2 Multi-Library Search

The following Functions allow an optional multi-library search during retrieval of input:

- a. COMPILE
- b. LINK
- c. EXECUTE
- d. MDCollect
- e. MDPRINT

A maximum of nine libraries may be searched. Each particular library is identified by a unique project/library pair of keywords.

By permitting a multi-library search, the PSL provides the capability of automatically mixing development code with well-tested code for compilation and execution without altering the contents of the individual libraries. A multi-library search by the management data collection and reporting Functions permits data to be collected and reported from multi-projects as well as multi-libraries.

3.3.3 Independent Files

The PSL system is designed to store card-image data in blocks in a random file. This card-image data is read by special PSL access routines when it is retrieved by modules in the PSL system. However, if the data must sometimes be read by programs outside the PSL system, the data must be stored in

181
5
a separate, or independent file. Examples of such data are unstructured code, which will be read by a standard compiler, and test data, which will be read by a user's program.

If a new unit is added to the SOURCE section for which the language is not a structured language supported by a pre-compiler in the PSL system, the system will create an independent sequential file for the data. Since an unstructured unit is not processed by a precompiler and INCLUDE statements are not resolved, the independent SOURCE unit must be a complete compilable set of code and may not be reduced to smaller units, as may be done with structured code.

Units in the OBJECT (which are output from a compiler or assembler) and LOAD (input to the Loader) sections are automatically created and maintained in independent files by the PSL system.

3.3.4 Spawned Jobs

Certain PSL Functions, called job spawning Functions, require the execution of independent programs such as precompilers, compilers or collectors, or user written programs as part of their processing. To invoke an independent program, the PSL function retrieves a pre-stored JCL procedure, modifies the JCL and writes the modified JCL on a temporary file. Subsequent Functions may append additional JCL on this file. Also, JCL may be added directly to this file by the user with the JCL Function. When the end of the PSL input is reached and all the PSL Functions have been processed, the added JCL is spawned, using the temporary file as its single input stream.

The job spawning Functions are COMPILE, LINK, EXECUTE, MDCOLLECT, MDPRI NT, and JCL. JCL procedures are provided with the PSL system for use with these Functions. The EXECUTE Function invokes a user's program using the JCL stored by the user in the JOB section of his library.

3.3.5 OPTION Keyword

A special keyword is reserved for those PSL Functions which can result in a spawned job activity but is presently implemented for the COMPILE Function, only. The OPTION keyword permits user designated options to be substituted for the default options specified in the JCL card flagged with the word "COMPILE" in columns 73-79.

3.3.6 Stub Generation

The PSL generates stubs for missing units under two conditions:

- a. When structured source code is added to a library, a dummy SOURCE unit (SOURCE stub) is provided for any unit which is referenced in an INCLUDE statement, but has not yet been added to the library. (See description of the INCLUDE statement under subsection 3.3.9). Accounting information for the source stub is stored in the SOURCE section of the user's library.
- b. When an object module is requested on an INCLUDE card in the user's loader control cards and the object module is not found by the PSL system, a dummy object module (OBJECT stub) is provided to the Loader by the system. A message will be printed on the system output file when the object stub is executed. The object stub is not stored in the user's library.

The object stub enables program execution with partially completed code without requiring the individual programmer to create special code which will subsequently be replaced.

3.3.7 Error Handling

When an error is encountered on a Function card, an error message is generated and the remaining keywords for that Function are scanned, but no library processing is done. Input is read and printed, but not processed, until the next Function card is found. If an error is found on a PARAM Function, no processing takes place until another PARAM Function card is found.

If an error occurs while processing a Function, the PSL system will attempt to undo whatever processing has taken place and will then terminate the Function. Advisory and error messages (see Reference B) will be generated. Processing will resume at the next Function card.

3.3.8 INCLUDE Statements

An INCLUDE statement is used in source code and with loader control cards to refer to a unit which will be retrieved and substituted in-line for the INCLUDE statement.

The INCLUDE statement may be used in the SOURCE, PDL, and LINK sections as follows:

- a. SOURCE - When the INCLUDE statement is used as a line of code in a SOURCE unit, it refers to a lower-level unit of SOURCE code which will be substituted in-line for the INCLUDE statement. Such INCLUDE statements are resolved by a pre-processor before the code is passed to the compiler. Therefore, INCLUDE statements may only be used in modules which are written in a structured language, such as SCOBOL or SPFORT, which are supported with preprocessors in the PSL system. Independent units may not be INCLUDED by other units and should not contain INCLUDE statements. When a non-independent unit is added to a SOURCE section, or modified, the INCLUDE references are checked. If an included unit does not exist in the section, a SOURCE-stub unit is created and is tagged with the name and language of the including-unit. When the including-unit is compiled, the

lower-level unit is a stub, an appropriate statement is inserted by the preprocessor to generate an output message when the code is executed. INCLUDE statements may be nested to a depth of 50 levels.

- b. PDL - The INCLUDE statement is processed in the PDL section in the same way as in the SOURCE section. A Program Structure report may be generated for a unit in the PDL section.
- c. OBJECT - When the INCLUDE statement is used with Loader control cards in the LINK section, it refers to a compiled unit (a module) in the OBJECT section which will be inserted into the input stream passed to the Loader. If the unit is not found in the libraries selected on the LINK (or EXECUTE) Function card, an OBJECT-stub unit is substituted in its place (with the unit-name as the external-name of the stub). This unit appears as a "STUB" in the load map of the user's program and a message will be printed when the OBJECT stub is executed.

3.3.9 Management Data Cycling

Data cycling is the periodic resetting of those management data items for which values have been accumulated over a period of time. Data cycling can be optionally defined to permit the user to specify the period for which values are to be accumulated before being reset.

"Cyclic" management data is first implemented in the Unit Accounting Record for PSL sections in which the MGMT=YES option is indicated. Two cyclic data items are maintained in that record: lines input per cycle and number of updates per cycle. The determination as to when the specified cycle period has elapsed is made when a management data collection is performed. If the number of days specified for the cycle period is equal to or exceeds the number of days elapsed since the accumulation of cycle statistics was begun, the cyclic data item contents will be reset to zero (i.e., recycled) after the collection of that data. A new data cycle is started and cycle duration is determined from that date for all SOURCE section units linked through the management plan; that is, through the module names listed in the management plan unit.

3.3.10 Management Data Archiving

Archiving enables the user to retain previously collected data for a historical review of project status. The determination to automatically archive is made when the data is collected,

based upon the current date, the start-of-cycle date, and the cycle duration. Actual archiving is not performed, however, until the next data collection is performed so that the most recent data collection remains the "current" data collection until such time as it is archived and/or replaced.

Each archived unit is uniquely identified at the time of archiving by affixing a suffix to its basic name. This suffix consists of a three digit serial number and a six character date. The assigned serial number (starting with 001) is increased by one for each subsequent collection archived. The six character date, given in MMDDYY format, signifies the day on which the data collection was performed.

Archives may be retained on disk or written to a backup tape. Tape-stored archives can be restored from tape for printing at some later time.

3.3.11 Transportability

The design and implementation of PSL permits transportability between DMA sites. To install and maintain PSL, **ASCTI** COBOL and ANSI FORTRAN compilers are required as part of the site inventory. If FORTRAN is not used at a site, it is possible to exclude the FORTRAN precompiler without impacting the remainder of PSL operation.

Only the PSL load modules are required to be disk-resident. After installation, both source and object modules may be saved on tape thus not impacting disk storage at a site. The size of the users' libraries are not a function of PSL, but rather are limited by the available storage at any particular site.

Section 4

PHASE II PSL ENHANCEMENTS

New and enhanced PSL functional capabilities were incorporated in Phase II of the DMA PSL conversion effort as follows:

1. CREATE - Create a Section

Within a library, the PSL system allows the user to create the following addition in the PSL section:

- o USER - This section contains and accounts for data that may be input to and output from standard PSL and/or special user operations. Data accessible to standard PSL operations is accounted for via the section index; data created and maintained by user programs only is accounted for via the section directory.

2. CHANGE - Change a Unit

An enhancement was developed and implemented to convert a random-stored PSL unit to an independent (sequentially-stored) unit when and if section space is exhausted during a unit update. The following advisory message is printed when such occurs:

ADV086 SECTION SPACE EXHAUSTED. UNIT TYPE
 CHANGED TO INDEPENDENT

In this case, the user may increase the space allocate to the specific section and restore the converted unit to its original random-stored unit type by performing the following:

- (a) use the BACKUP Function to save the section at which time the converted unit is written to the backup file as if it were its original unit type.
- (b) use the TERMINATE Function to release the existing section.
- (c) use the CREATE Function to create a larger section space.

- (d) use the RESTORE Function to restore the contents of the section from the backup file written in the previous operation.

3. PERFORM - Perform Job Control Cards

The PERFORM Function is primarily used to interface data between PSL and non-PSL operations. Job control cards contained in a user-designated JOB section unit are output to the spawned job file. These job control (e.g., \$FILE) cards in which the user has specified a flagword (starting in column 73), are processed to create independent unit files and update related PSL section accounting information as well as output \$PRMFL cards with the necessary FMS catalog/file information to control input/output operations that access the independent files.

4. PRECOMPILE - Precompile a Module

The PRECOMPILE Function is used to read the top and INCLUDED units of a source code module and produce an output containing the total module source code. This output may subsequently be directed to PSL storage or a user-designated program procedure.

5. COMPILE - Compile a Module

The COMPILE Function is enhanced to accommodate the compilation of JOVIAL language source code. Added keyword inputs are accepted to provide for the storage of COMPOOL symbolic tables and their reference in main program compilations. Structured JOVIAL code is preprocessed to produce code acceptable to the JOVIAL (i.e., JOCIT) compiler. The COMPILE Function is additionally enhanced to preprocess non-structured COBOL, FORTRAN, JOVIAL and GMAP source modules from PSL random-block storage to collect included code for input to the compilation activity.

6. SOURCE - Print a Card-Image Unit

The SOURCE Function is enhanced to print structured JOVIAL source code in an indented format and to print structured figure error messages when errors are detected in the structured JOVIAL unit code.

7. PURGE - Purge a Unit

The PURGE Function has been expanded to delete files listed in the section directory (primarily of the USER section). The keyword FILE is used to denote the keyword value naming the file to be purged.

Section 5

PSL SYSTEM TESTS

DMA PSL software deliveries were preceded by a system test at DMAHTC, Washington, D. C. and DMAAC, St. Louis, Mo. which demonstrated the PSL capabilities being delivered.

Appropriate test data were stored in the delivered system program file. Listings of these data were included as an adjunct to the Test Plan.

5.1 DMAHTC System Test

A PSL system test was conducted at DMAHTC in a two day period beginning 4 May 1978 and concluding 5 May 1978. Due to computer outage problems in that time period the review of system test results was mainly directed to results produced in the pre-system test checkout performed by IBM on 3 May 1978. To conclude the system test review, a demonstration of the DMA PSL system installation procedure was given on 5 May 1978 in which the PSL system project library was initialized with a user-selected project ID. The JCL procedures required for PSL system operation were stored in the JOB section of the PSL system project library and a listing of the stored procedures was printed under PSL system program control.

5.2 DMAAC System Test

A PSL system installation and test was performed at DMAAC in a two day period beginning 10 May 1978 and concluding 11 May 1978. Difficulty was immediately experienced when the DMAAC UNIVAC system-generated limit of four (4) alternate symbiont files was exceeded. Since at least five (5) alternate symbiont files are required by the DMA PSL system, this limitation presented a problem which evidently does not occur at the DMAHTC location due to a higher limit being set in their UNIVAC system generation procedure.

The problem was temporarily resolved by reducing the PSL alternate symbiont requirement to four (4) files through redefining the ACFL9, ITRDE, ITWRE and PPLKE programs to reference a mass storage file for PSL-independent units rather than an alternate symbiont file. The four programs

187
0

were recompiled and a new batch control (BCTL) load (i.e., absolute) element was produced using standard DMA PSL maintenance procedures. The PSL system project library was then initialized with JCL procedures, as required.

System testing was then conducted to demonstrate that the PSL function and subfunction capabilities were operable and produced acceptable data results. At the conclusion of the system test, a request was made by DMAAC to re-attempt PSL operation with five (5) alternate symbiont files, subsequent to an UNIVAC system "patch" being made to raise the previous limit. The delivered DMA PSL system BTCL load element was restored and a test was made using PSL input directives to add an independent unit. Results of the test were successful in that no abort occurred and the added unit data lines were accessed by the UNIVAC Edit processor.

The higher alternate-symbiont limit should be retained in order that PSL-independent unit data can be externally accessed and/or input, especially since this capability is a designated feature of the PSL system. System test operations were concluded at this point with no outstanding deficiencies being noted.

Section 6

PSL DELIVERIES

The following items were delivered:

- a. Users Manual
- b. Operations Manual
- c. Program Maintenance Manual
- d. PSL Software System
- e. System Test Results
- f. Technical Report